面向实时事件流的复杂事件处理方法 *

邱 涛 [1]，谢沛良 [1†]，邓国鹏 [2]，郗红梅 [2]，郑 智 [2]，夏秀峰 [1]

(1. 沈阳航空航天大学 计算机学院，沈阳 110136; 2. 沈阳飞机工业(集团)有限公司 试飞站/试飞实验室，沈阳 110034)

摘 要：复杂事件处理是一种动态环境下对事件流进行分析的技术。复杂事件处理技术通常基于有限状态自动机实现，匹配过程中会在事件流上产生大量且重叠的部分匹配，有限状态自动机需维护大量的重复匹配状态，导致基于该技术的方法都会出现冗余计算的问题。为了提高复杂事件处理的匹配效率，提出了使用复杂事件实例覆盖技术来实现复杂事件处理的方法。通过设计临时匹配链式分区存储结构以及基于此结构的匹配算法，来利用复杂事件实例覆盖减少冗余计算，从而实现匹配效率的提升。在模拟数据集和真实数据集上进行了实验测试与分析，与两种常用的复杂事件处理技术进行比较。实验表明，提出的方法能够在保证匹配正确性的同时有效地减少匹配过程中的冗余计算，提高整体匹配效率。

关键词：复杂事件处理；查询优化；有限状态自动机；分区存储

中图分类号：TP315　　doi: 10.19734/j.issn.1001-3695.2021.12.0707

Complex event processing method over real-time event streams

Qiu Tao[1], Xie Peiliang[1†], Deng Guopeng[2], Xi Hongmei[2], Zheng Zhi[2], Xia Xiufeng[1]

(1. *Dept. of Computer*, *School of Shenyang Aerospace University*, *Shenyang* 110136, *China*; 2. *Flight Test Station of Shenyang Aircraft Industry* (*Group*) *Co. LTD*, *Shenyang* 110034, *China*)

Abstract: Complex event processing is a technology for analyzing the event streams in a dynamic environment. Complex event processing technology is usually implemented based on finite state automaton. During the matching process, a large number of overlapping partial matches will be generated by the event stream. The finite state automaton needs to maintain a large number of repeated matching states, which leads to the problem of redundant calculation in the methods based on this technology. In order to improve the matching efficiency of complex event processing, this paper proposes a method of using complex event instance coverage technology to realize complex event processing. By designing a temporary matching chain partition storage structure and matching algorithms based on this structure, redundant calculations can be reduced using complex event instance coverage, thereby achieving an improvement in matching efficiency. Experiments are performed on simulated and real datasets, and compared with two commonly used complex event processing technologies. The experimental results show that the proposed method can effectively reduce the redundant computation in the matching process while ensuring the correctness of the matching, and improve the overall matching efficiency.

Key words: complex event processing; query optimization; nondeterministic finite automaton; partition storage

0 引言

随着信息社会的进一步发展，越来越多的行业采用复杂事件处理(complex event processing, CEP)技术来对海量的事件流进行实时的分析。复杂事件处理通过分析事件中的关系，通过关联、聚合、过滤等技术，根据事件间的时序关系和聚合关系制定查询规则，持续地从事件流中获取符合要求的事件序列。该技术在金融交易分析[1,2]、传感器网络[3]、物联网[4~6]、交通[7]领域都有着广泛的应用。

目前，基于有限状态自动机(nondeterministic finite automaton, NFA)的处理模型是最流行的复杂事件处理技术实现方式，例如 SASE[8,9]、Cayuga[10,11]和 Siddhi[12]。通过 NFA 的方式来实现的复杂事件处理技术，在事件流上进行匹配的过程中都会产生临时匹配，所产生的临时匹配能够被后续的事件使用，并且生成新的临时匹配以及最终的匹配结果。所以匹配过程中会在事件流上产生大量且重叠的部分匹配，有限状态自动机需维护大量的重复匹配状态，导致基于该技术的方法都会出现冗余计算的问题。尤其当复杂事件查询的时间窗口跨度大时，冗余计算将会给处理器和存储器等硬件资源带来巨大的额外开销。

为了减少基于有限状态自动机的复杂事件处理技术中的冗余计算，提高匹配效率，本文利用链式分区存储的结构来管理临时匹配，利用复杂事件实例覆盖来减少临时匹配的冗余产生和冗余拷贝，从而提高复杂事件匹配的效率。

综上所述，本文的主要贡献如下：

1) 提出了复杂事件实例覆盖的概念，通过复杂事件实例覆盖，可以建立临时匹配之间的关联关系，基于此关系来减少临时匹配的冗余产生和冗余复制。

2) 设计了一个临时匹配链式分区存储结构，通过该结构避免了临时匹配的集中式存储及使用，同时该结构也作为复杂事件实例覆盖的载体，构建了临时匹配之间的关联。

3) 提出了基于临时匹配链式分区存储结构的复杂事件

收稿日期：2021-12-30；**修回日期：**2022-03-28　**基金项目：**国家自然科学基金青年基金项目(62002245)；辽宁省教育厅基础研究项目(JYT2020027)

作者简介：邱涛(1989-)，男，江西萍乡人，副教授，硕导，博士，主要研究方向为序列数据处理、查询优化、数据挖掘；谢沛良(1997-)，男(通信作者)，江西赣州人，硕士研究生，主要研究方向为复杂事件处理(xiepl1997@163.com)；邓国鹏(1977-)，男，辽宁开原人，学士，主要研究方向为遥测数据分析处理；郗红梅(1969-)，女，辽宁沈阳人，学士，主要研究方向为遥测数据分析处理；郑智(1988-)，男，辽宁抚顺人，学士，主要研究方向为遥测数据分析处理；夏秀峰(1964-)，男，山东青岛人，教授，硕导，博士，主要研究方向为数据库、数据仓库、数据挖掘.

匹配算法 CoverMatch 和 CombineMatch。通过这两个算法，能够在减少临时匹配数量和复制的情况下保证复杂事件处理匹配结果的正确性和完整性。

4) 通过在模拟数据集和真实数据集上对优化方法的性能进行对比实验和分析，验证了所提方法和算法的有效性和高效性。

1 相关工作

复杂事件处理是从事件驱动业务出发的，将系统中产生的每一条数据记录都看成是一个事件，实时输入的数据流即为实时事件流，复杂事件执行引擎会根据事先制定好的复杂事件描述规则，来对事件流进行相应的判断、过滤、关联等操作，然后给用户输出一系列的更高层次的复合事件。其中复杂事件描述规则一般包含用户感兴趣的事件语义，或者是专业领域中某种既定的标准和规范。也就是说，复杂事件处理能够在实时事件流中识别某一个人为定义的复合事件，并且为用户反馈识别的结果。

目前复杂事件处理技术已经有了很多的研究成果，复杂事件处理技术一般采用非确定性有限状态自动机的变体模型来处理复杂事件的识别。

Diao 等人提出的 SASE 是一种复杂事件处理引擎，同时还提出了一种能够定义复合事件的事件描述语言 CEL[13, 14]，这种语言具有类似 SQL 的高级结构，可定义事件序列、匹配策略、事件约束和事件窗口约束等。通过 SASE 引擎可将事件描述语言所定义的复合事件转变为非确定性有限状态自动机，从而实现在事件流上的事件获取和计算。Diao 等还在 SASE+[15]中对 SASE 进行了扩展，引入了对克林闭包、否定和聚合操作的支持。SASE/SASE+主要的缺陷在于，在 NFA 进行匹配的过程中，需要产生匹配结果的临时匹配，同时为了保证结果的准确性，不允许在时间窗口约束内将临时匹配丢弃，这种机制导致了临时匹配的堆积，从而影响了自动机的处理效率。

康奈尔大学开发的 Cayuga 也采用 NFA 作为计算模型来处理事件的识别，但是它对于事件的描述能力相对较差。Cayuga 支持发布-订阅技术，并且提供了良好的可扩展性，此外还运用了查询优化技术，可将多个拥有相同时间戳的具有等价状态的事件一同进行处理。但是由于它的内核是单线程的，并没有有效地从这些优化技术中获益。

FlinkCEP[16]在设计思想上和 SASE 很接近，同样使用事件约束作为 NFA 节点状态转变的条件。从事件描述语言的角度来看，FlinkCEP 与 SASE 的一个最大的区别是 FlinkCEP 没有支持定义复合事件的语言。为了替代事件描述语言，FlinkCEP 需要用户使用 Java 或者 Scala 编写事件描述，这种定义方式可读性差且编写时易出错。

除了基于 NFA 模型实现的复杂事件处理技术之外，另外一种使用树来作为计算模型的实现复杂事件处理的技术也得到了广泛的研究和应用。

Mei 等人提出的 ZStream[17]是 CEP 领域基于树实现复杂事件处理技术的典型研究成果。ZStream 的事件描述语言与 SASE 非常相似，所遵循的语法大多数都相同。ZStream 将事件存储在叶子节点，内部节点对应于操作符。在进行事件流处理时，它不会立刻判断到达事件的约束条件，而是先把事件进行收集到一定量，进而对其进行批处理。树结构和批处理的结合允许 ZStream 根据期望成本和条件约束来执行各种复杂事件处理任务。例如，对于给定的复杂事件序列<*A*, *B*>，SASE 将会为每一个出现的 *A* 事件新建一个临时匹配，即使 *B* 事件的出现概率很低，而 ZStream 则可以遵循另一个匹配规则，一直等待 *B* 事件的到达，再去批量检查先前到达的 *A* 事件。但是 ZStream 仍然不能避免大量的未处理事件的堆积，这和 NFA 产生临时匹配的堆积本质上是一样的。

基于以上的研究可以发现，无论是基于 NFA 还是基于树的方法来实现复杂事件处理技术，为了复杂事件匹配结果的正确性和完整性，都需要存储至少一个时间窗口范围内的临时匹配。假设时间窗口跨度很大，复杂事件的处理将对处理器计算能力和存储器等硬件资源都带来了不小的负荷。

2 预备工作

复杂事件处理是一种面向事件流的查询分析技术，其目标是从大量的基本事件构成的事件流中，找出满足复杂事件描述语义的更高层次的复杂事件。本章节将详细介绍复杂事件处理涉及的预备知识。

定义 1 事件流。事件流 $S(s_1,s_2,\cdots,s_n)$由一系列的基本事件实例构成，其中 s_i 为事件实例，它包含了事件的类型、事件的属性和事件发生时的时间戳等信息。

事件流往往是由多个数据源的数据构成的，在使用海量数据的很多研究应用领域里，例如气象预测[18]、海上航行[19]和交通数据研究[20]，数据源可以是采集设备或者是传感器，因此需要先对数据使用融合技术[21, 22]进行融合以得到包含多事件类型的事件流。

图 1 展示的是船只在航行中产生事件流的示例。其中事件流包含了 *A*、*B*、*C* 三种事件类型，*A* 表示低速启动，*B* 表示船头左转 90°，*C* 表示低速停靠。每个事件实例都包含时间戳以及属性值，此处使用相应的小写英文字母表示事件实例。例如，b_1 为 *B* 事件类型的第一个事件实例，其时间戳为 2，此外还包括行驶速度、行驶方向以及倾斜角度等属性。

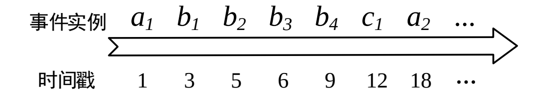


图 1 事件流示意图

Fig. 1 An example of events stream

通过观察图 1 所示的事件流可以得知，在时间窗口 1~12 内，船只先是低速开始启动，之后在航行过程中转了一个圈，最后低速停靠。

复杂事件是由事件流 $S(s_1,s_2,\ldots,s_n)$上的若干事件实例构成的一个复合事件，表示为 $R(r_1,r_2,\ldots,r_n)$，$\forall r_i \in S,(0 \le i \le n)$ 。复杂事件表示在事件流上发生的客观存在的具体事件，其语义通常通过复杂事件描述语言所定义的查询来进行表示。

定义 2 复杂事件查询。复杂事件查询 *Q* 是由一组定义在基本事件上的约束条件构成，用以定义和表示更高层次的复杂事件的属性特征。

当前的研究工作中已提出多种形式的复杂事件描述语言用以定义复杂事件查询，其中 SASE 提出了最具有代表性的复杂事件描述语言。其具备简洁的语法规则，灵活的表达能力，所以本文将使用 SASE 复杂事件描述语言来定义复杂事件查询。SASE 事件描述语言是一种声明式语言，总体结构如下：

```
PATTERN   <sequence pattern>
[WHERE    <qualification>]
[AND      <other qualifications>]
[WITHIN   <time window>]
```

使用上述的结构可以编写复杂事件查询。默认情况下，查询将读取事件流中实时到达的事件，从而进行复杂事件处理，最后将匹配成功的复杂事件反馈给用户。

为了解释 SASE 事件描述语言结构的含义，现在使用一个基于道路交通场景构造的例子来说明。在这个例子中，事

件类型 TrafficInfo 为道路地点所收集到的该地点的交通数据报告，一个报告对应一个事件实例。假设报告内容包含该地点的位置，以及某一时刻的车流量、平均车速等。所构造的查询如 Q_1 所示。

```
Q1 :
PATTERN   SEQ(TrafficInfo a, TrafficInfo+ b[])
WHERE     skip-till-any-match
AND       a.pos = tollA
AND       b.pos > a.pos + 5 miles
AND       b[i].count > b[i - 1].count
WITHIN    1 hour
```

上述查询定义了一个复杂事件：距离 A 收费站 5 英里远的某地点，其车流量在 1 小时的时间范围内逐渐增大，其中事件之间的默认时间戳约束为 *a.timestamp*<*b.timestamp*、*b[i].timestamp*<*b[i+1].timestamp*。其中 PATTERN 部分定义了复杂事件的事件序列，使用 SEQ 结构来指定两个事件类型构成事件序列。可以看到两个事件类型都是 TrafficInfo，后者使用了克林闭包，使用“+”来表示一个或者多个指定类型的事件，同时需要结合“[]”来申明。除了定义顺序序列和闭包序列，PATTERN 部分还可以定义否定操作，只需要在事件类型前面加上“!”，例如 *SEQ(A a, !B b, C c)*表示在 *a.timestamp*<*c.timestamp* 的条件下，*A* 类型的事件实例和 *C* 类型的事件实例之间不允许出现 *B* 类型的事件实例。

WHERE 部分指定了当前查询所用到的匹配策略，skip-till-any-match 表示将在事件流中匹配所有的结果，本文将只讨论该策略下的复杂事件匹配，其他策略匹配出的结果都是 *skip-till-any-match* 策略所匹配结果的子集，故不做另外讨论。AND 部分定义了事件约束，是作为 WHERE 约束的延伸。WITHIN 则定义了时间窗口约束，将所要匹配的结果事件跨度限制在一定范围内。

通过以上事件描述语言的语法规则写成的一个复杂事件查询 *Q*，将通过复杂事件处理引擎在事件流上进行查询分析，并得到查询结果。

3 查询优化技术

本章将对本文提出的复杂事件查询优化技术进行详细的介绍。首先对于基于有限状态自动机的复杂事件匹配技术进行分析，通过设计并实现一个临时匹配链式分区存储结构以及查询优化算法，来优化基于有限状态自动机的复杂事件匹配方法的技术，提升查询匹配的效率。

3.1 基于有限状态自动机的匹配方法

基于有限状态自动机的匹配方式是目前使用最广泛并且有效的复杂事件匹配技术。以 SASE 为例，处理一个复杂事件查询的步骤有：(1) 将查询解析为一个有限状态自动机；(2) 读取事件流；(3) 进行复杂事件匹配，产生临时匹配结果或成功匹配结果。

本节通过构造一个查询 Q_2 及其匹配过程来详细介绍基于有限状态自动机的匹配方法。

```
Q2 :
PATTERN   SEQ(A a, B+ b[], C c)
WHERE     skip-till-any-match
AND       b[i].val >= b[i+1].val
WITHIN    50
```

Q_2 包含了一个 *B* 事件类型的克林闭包，表示匹配一个或者多个 *B* 类型的事件实例，并且在 *B* 事件的约束条件下，Q_2 只会匹配属性值 *val* 递减的 *B* 事件序列，所以 Q_2 匹配的复杂事件是以 *A* 事件类型的事件实例开头，然后是 *B* 事件类型的 *val* 递减事件实例序列，最后以 *C* 事件类型的事件实例作为结尾。将 Q_2 以文本的形式作为 SASE 的输入，Q_2 经过编译后，在 SASE 内部得到一个对应的有限状态自动机，如图 2 所示。

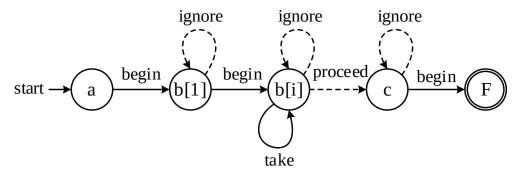


图 2 Q2 对应的有限状态自动机

Fig. 2 The NFA of Q2

在图 2 展示的有限状态自动机中，begin 表示在当前状态下获取一个符合条件的事件实例并保存，从而转移到下一个状态。ignore 表示当遇到不符合条件的事件实例，就保持当前状态不变。take 和 proceed 是闭包节点独有的动作，take 表示获取相同类型并且符合条件的事件实例，并保持当前状态。而 proceed 可以将 NFA 从闭包节点状态跳出到下一个状态，并且 take 和 proceed 动作是同时发生的。也就是说在 *b[i]* 节点，状态机的状态既可以转换为 *c*，也可以保持 *b[i]*的状态不变。

在 SASE 中是通过 NFA 来处理事件流的，例如当数据流中来了一个 *A* 事件类型的实例，则图 2 中的 NFA 则会初始化一个包含该实例的临时匹配，这个临时匹配会一直存在，直到事件流当前的时间戳与该临时匹配的时间跨度超过了查询的时间窗口，才会把其移除。此外，当 NFA 在该临时匹配的基础上与当前事件流中的事件能够发生状态变化时，将会对临时匹配进行拷贝，然后把新的事件添加到新拷贝的临时匹配上，而原来的临时匹配继续保留。这样做的目的为的是保留 NFA 的上一个状态以及其中的匹配序列。

为清晰说明基于有限状态自动机的匹配过程，假设 Q_2 在一段特定事件流 $S(a_1,b_1,b_2,b_3,c_1)$ 上进行查询，事件流 *S* 的时间戳和属性值如图 3 所示。

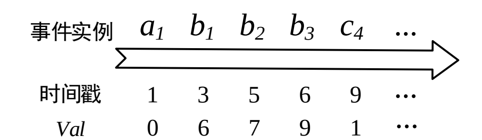


图 3 事件流 S 示意图

Fig. 3 The details of events stream S

Q_2 在 *S* 上的匹配过程，可以通过图 4 来展示。

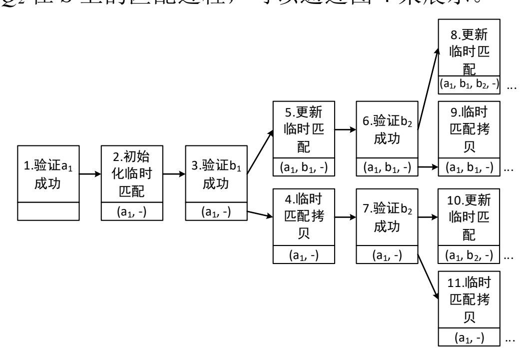


图 4 查询 Q2 在事件流 S 上的匹配过程

Fig. 4 The matching process of Q2 on the event stream S

由图 4 可知，在 SASE 中通过有限状态自动机来处理事件流，当 a_1 事件到达时，验证成功，初始化一个包含 a_1 事件实例的临时匹配(a_1,-)，该临时匹配的存在表示着图 2 中 NFA 的“*a*”节点状态的匹配存在。当 b_1 事件实例到达并且验证成功后，系统将会把(a_1,-)备份，然后将原有的临时匹配进行更新，更新为(a_1,b_1,-)，所以系统中出现了两个临时匹配，等到下一个到达的事件并且验证成功后，就需要对两个临时匹

配进行拷贝和更新，于是就出现了 4 个临时匹配。

通过这个方法显然可以将事件流中的正确匹配结果都找出来，但是缺点也是显而易见的，在匹配的过程中产生了很多的临时匹配用以表示临时匹配序列，同时用以保存 NFA 的匹配状态。所以随着匹配的进行，最坏的情况下系统中的临时匹配数量呈现指数增长趋势，系统的计算开销和存储开销也将越来越大。

3.2　减少冗余计算

针对基于 NFA 的复杂事件处理中产生的大量临时匹配和拷贝而导致的冗余计算，本文提出了“事件实例覆盖”和“复杂事件实例覆盖”的概念，同时设计了一个临时匹配链式分区存储结构和基于此结构的匹配算法 CoverMatch 来减少冗余计算从而提高基于 NFA 的复杂事件处理的性能，另外设计了匹配结果联结算法 CombineMatch 来完善匹配结果的生成。

定义 3　事件实例覆盖。当事件实例 s_i 与事件流中相邻的前一个事件实例 s_{i-1} 同属一个事件类型，并且都能够被当前的 NFA 验证成功且作用在 NFA 的同一个状态节点，则称 s_i 是 s_{i-1} 的一个事件实例覆盖。

定义 4　复杂事件实例覆盖。对于两个匹配结果 M_1 和 M_2 来说，如果 M_1 中的每一个事件实例，都相等于 M_2 中对应的事件实例或是 M_2 中对应事件实例的事件实例覆盖，则称 M_1 是 M_2 的一个复杂事件实例覆盖。

事件实例覆盖是可传递的，例如在图 3 的事件流 S 中的三个事件实例 b_1、b_2、b_3，事件类型都是相同的，并且能够被 Q_2 对应的 NFA 验证成功且作用在同一个状态节点，所以 b_2 是 b_1 的事件实例覆盖，b_3 是 b_2 的事件实例覆盖，则 b_3 也是 b_1 的事件实例覆盖。同理，复杂事件实例覆盖也是可传递的。例如在图 3 中的事件流上，进行 $SEQ(A\ a, B\ b, C\ c)$的匹配，则会得到的匹配结果是 $m_1(a_1, b_1, c_4)$、$m_2(a_1, b_2, c_4)$、$m_3(a_1, b_3, c_4)$。按照定义 4，m_2 是 m_1 的复杂事件实例覆盖，m_3 是 m_2 的复杂事件实例覆盖，且可传递，则 m_3 也是 m_1 的一个复杂事件实例覆盖。

很多场景都能够应用复杂事件实例覆盖，因为复杂事件实例覆盖能运用在匹配距离当前事件最近的复杂事件上，例如在股票事件流上寻找某一只股票最近的一次价格反弹事件、在交通数据事件流上寻找某地点最新的拥堵事件等等。

在复杂事件匹配的过程中，如果只去匹配复杂事件实例覆盖的话，在出现连续的相同事件类型实例的情况下，可以以更快的效率完成匹配任务。同时，相对于 SASE 中的基于 NFA 下的 *skip-till-any-match* 利用大量临时匹配计算所有结果的策略，使用本文提出的方法可以在匹配复杂事件实例覆盖的同时，利用结果联结算法，可以得到所有的匹配结果，过程高效并且不产生额外临时匹配，从而达到减少冗余计算的目的。

3.2.1 临时匹配链式分区存储结构

为了让有限状态自动机支持本文提出的优化方法达到减少冗余计算的目的，本文对 NFA 进行了增强，为其设计了一个临时匹配的链式分区存储结构，用以替代原来的集中式的临时匹配存储方式。集中式的临时匹配存储方式的缺陷上面已经提到过了，每次新的事件到来时都需要对所有的临时匹配进行遍历验证，这将严重消耗计算资源。而临时匹配链式分区存储结构则借助事件实例覆盖和复杂事件实例覆盖的概念来设计，规避了集中式临时匹配存储的缺点。

假设当前存在查询 Q_4：$SEQ(A\ a, B\ b, C\ c)$，以及事件流 $S_t(a_1,a_2,b_1,b_2,a_3,b_3,c_1,c_2)$。为了简明易懂，$Q_4$ 的事件约束条件将置为空，且设置其时间窗口大于事件流 S_t 的时间范围。当查询 Q_4 在事件流 S_t 上进行复杂事件实例覆盖匹配时，临时匹配链式分区存储结构示意图如图 5 所示。

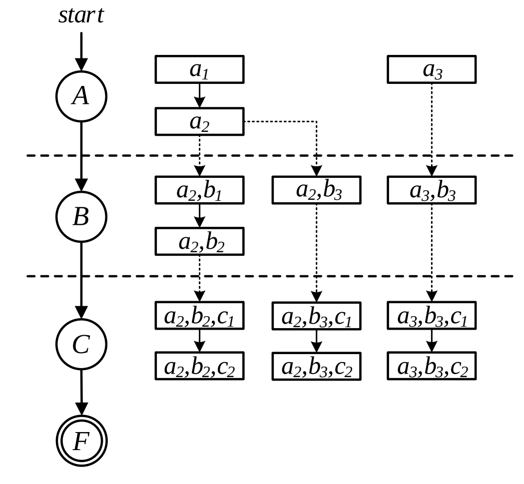


图 5　临时匹配链式分区存储结构

Fig. 5　Temporary matching chain partition storage structure

在 Q_4 被编译成 NFA 之后，存在三个主要状态(状态 F 为终止状态)，这三个状态将分别产生三个分区，简称为 A 分区、B 分区和 C 分区。事件流 S_t 中的事件实例 a_1 和 a_2 到达后处理系统之后，将分别产生临时匹配并存储在 A 分区，同时分别打包成链表节点<a_1>和<a_2>。由于 a_2 是 a_1 的事件实例覆盖，所以<a_2>将作为<a_1>的后继节点，并且暴露给下一个 B 分区。在一个分区中所有的链表都仅将链表尾节点暴露给下一个分区。当事件实例 b_1 到达，首先会在上一个 A 分区中寻找暴露节点并拷贝下来，也就是<a_2>节点，然后进行更新成为<a_2, b_1>节点，并作为链表头存储在 B 分区并暴露给 C 分区。随后事件实例 b_2 到达，b_2 也会找到 A 分区的暴露节点<a_2>，会产生一个节点<a_2, b_2>，由于<a_2, b_2>是<a_2, b_1>的复杂事件实例覆盖，所以<a_2, b_2>会成为<a_2, b_1>的后继节点，并且替代<a_2, b_1>暴露给 C 分区。同理，当事件流 S_t 中的事件都到达并被处理之后，最终会得到 3 个复杂事件实例覆盖<a_2, b_2, c_2>、<a_2, b_3, c_2>、<a_3, b_3, c_2>。该匹配过程见算法 1。

算法 1　基于链式分区存储结构的匹配算法 CoverMatch

```
输入：事件流 Se，查询 Q。
输出：复杂事件实例覆盖集 Rs。
a) e←null;
b) Rs←Ø;
c) nfa←parse(Q);
d) while (e←Se.nextEvent()&&e!=null) do
e)   if !nfa.verfy(e) do
f)     continue;
g)   tempMatchList←buildTempMatch(e);
h)   checkTimeWindow(tempMatchList);
i)   if tempMatchList.isEmpty() do
j)     continue;
k)   region←nfa.getRegion(e.eventType);
l)   for each tempMatch in tempMatchList do
m)                  node←getDominateMatch(tempMathch,
region.getCandidates());
n)     if node!=null do
o)       node.next = tempMatch;
p)     else
q)       region.setCandidate(tempMatch);
r) for each r in nfa.getLastRegion().getCandidates do
s)   Rs.add(r.prev);
t) return Rs;
```

在算法 1 中，第 g 行的 buildTempMatch 方法表示了使用

事件 e 在上一个分区中遍历暴露的匹配节点，将能够进行更新的节点拷贝，进行更新，然后再进入到当前分区进行复杂事件实例覆盖的检查，如果不是某个节点的复杂事件实例覆盖，就单独成链(第 q 行)，否则成为某个链的链尾节点(第 o 行)。最后返回最终的复杂事件实例覆盖。由于采用的数据结构是双向环形链式结构，所以第 s 行使用 prev 可以直接定位到链尾节点从而得到最终复杂事件实例覆盖。

现在对算法 1 的时间复杂度和空间复杂度进行分析。若系统正在对事件 e 进行处理，假设待处理的事件流 S 中，在两倍的时间窗口内单一事件类型的实例数量最多为 K 个，则 e 事件所属分区中的链式结构实例数量最大值为 K，设事件 e 在所属分区中产生的临时匹配数量为 m，由于临时匹配的产生只根据上一个分区中每一个链结构的链尾节点来产生的，所以 m 的最大值也为 K，则处理 e 事件匹配的时间复杂度为 $O(K^2)$，算法 1 的时间复杂度为 $O(|S|K^2)$，其中$|S|$为事件流 S 中的事件个数。由于在事件 e 所在的分区的临时匹配链式结构中，只有尾节点会参与构建新的临时匹配节点，因此通过事件 e 得到新的临时匹配节点的空间复杂度为 $O(K^2)$。

在复杂事件处理中，临时匹配的拷贝处理需要使用深拷贝来实现。完成一次临时匹配的深拷贝需要在内存中产生一个新的临时匹配实例，并且将原临时匹配实例中的全部属性值以及实例中存在的引用，复制到新的临时匹配实例中。传统方法中临时匹配的拷贝是在遍历所有临时匹配的过程中进行的，这将不利于为复杂事件处理提供良好的系统吞吐能力，系统的计算开销也将增大。同时，所有临时匹配都被找出并显式地在内存中以实例的形式存在，将增加内存的开销。以查询 Q_4 在事件流 S_t 上进行匹配的过程为例，使用基于有限状态自动机的传统匹配方法进行匹配，需要产生 24 个临时匹配以及 21 次临时匹配的拷贝，而通过临时匹配链式分区存储结构以及上述匹配算法，只产生 13 个临时匹配以及 10 次临时匹配的拷贝。临时匹配的数量和拷贝次数以及遍历操作的减少将提高系统的吞吐能力，并减少内存的开销。

临时匹配链式分区存储结构是基于复杂事件实例覆盖的概念提出的。借助分区存储的特点，处理复杂事件时并不需要遍历系统中所有的临时匹配，在实现减少临时匹配数量和拷贝的同时，并不会对最后的正确匹配结果有所影响。

当用户仅需要距离当前时间最近的一个复杂事件实例，通过指定系统保存<a_3, b_3, c_2>即可。假设用户需要的是像 *skip-till-any-match* 策略那样将所有的结果全部匹配出来，则需要使用到所有最终的复杂事件实例覆盖，利用其所在链表上的其他节点信息来进行结果联结，从而得到所有的匹配结果。

3.2.2 匹配结果联结

本文进一步地完善了匹配结果的生成。当用户指定需要获取全部匹配结果时，则需要使用算法 1 所得到的结果来进行反向的匹配结果联结。继续使用查询 Q_4：$SEQ(A\ a, B\ b, C\ c)$，以及事件流 $S_t(a_1,a_2,b_1,b_2,a_3,b_3,c_1,c_2)$来举例，在图 5 所示的匹配过程及结果中，$Q_4$ 的查询结果是<a_2, b_2, c_2>、<a_2, b_3, c_2>、<a_3, b_3, c_2>，将通过该结果以及链式分区结构来得到全部的匹配结果。

由于采用的是双向环形链式结构，可以从最底端的复杂事件实例覆盖出发，反向遍历所在的链表。通过收集每个分区的该链表节点的最后一个事件实例，最后利用递归实现各个分区事件实例的联结，该过程如算法 2 所示。对于 Q_4 来说，只需要将<a_2, b_2, c_2>、<a_2, b_3, c_2>、<a_3, b_3, c_2>三个节点分别传入到算法 2 中，就能得到 Q_4 所有的匹配结果。

算法 2　匹配结果联结算法 CombineMatch

输入：复杂事件实例覆盖 *R*。

输出：所有匹配结果 *M*。

```
a) M←∅;
b) list←∅;
c) while true do
d)   list.add(R.getLastEvent());
e)   R←R.prev;
f)   if R.isHead() do
g)     R←getPrevBlockNode(R);
h)     break;
i) return M←doCombine(list, CombineMatch(R));
```

在算法 2 第 g 行的方法 getPrevBlockNode 中，通过传入链表顶点 R，能够获取上一个分区中 R 的传递节点。第 i 行使用了递归的形式来获取最终的联结结果，其中 doCombine 方法进行了联结操作。

现在对算法 2 的时间复杂度和空间复杂度进行分析。假设当前分区数为 n，假设在多复杂事件查询中所定义的最长的复杂事件序列长度为 N，则 n 的最大值即为 N，且通过递归调用函数次数最多为 N 次，若每个分区中与复杂事件实例覆盖 S 关联的临时匹配链平均长度为 m，则算法 2 的空间复杂度为 $O(Nm)$。由于每次递归函数体中的执行操作复杂度为 $O(m)$，因此算法 2 的时间复杂度为 $O(Nm)$。

利用临时匹配链上的节点信息进行结果联结，可在算法 1 的基础上得到全部的匹配结果。联结过程中无须进行临时匹配的构建和拷贝工作，并不会增大系统中临时匹配的规模，因此节省了系统的内存开销。

使用图 5 中最左侧的链表来举例，首先拿到 C 分区的节点<a_2, b_2, c_1>，在 C 分区通过获取每个节点的最后一个事件实例从而得到[c_1, c_2]，然后在 B 分区对应的链表获得了[b_1, b_2]，继而在 A 分区获得了[a_1, a_2]，通过递归 doCombine 方法将三者联结，得到八个匹配结果(a_1, b_1, c_1)(a_1, b_1, c_2)(a_1, b_2, c_1)(a_1, b_2, c_2)(a_2, b_1, c_1)(a_2, b_1, c_2)(a_2, b_2, c_1)(a_2, b_2, c_2)，同理，使用 C 分区的另外两个复杂事件实例覆盖也能得到其对应的所有匹配，最终通过此方式得到所有的匹配结果以实现 *skip-till-any-match* 策略的匹配效果，且避免了大量临时匹配的生成和拷贝。

4　实验

本章通过实验对比来对本文提出的基于临时匹配链式分区存储结构的匹配优化方法的有效性进行了分析和验证。实验采用 Java 实现了优化方法的编写。本章将从多个维度对实验结果进行分析说明，并通过性能对比证明其有效性。

4.1　实验设置

本文在两个数据集上进行了实验，其中第一个数据集是通过事件流生成器所生成的模拟数据流，第二个数据集是真实数据集。

第一个数据集是 ABC 类型事件流，该数据集通过把事件类型简写为大写英文字母方式来定义事件的类型。其中每个事件都带有时间戳以及各自的属性值。生成事件流之前可在事件流生成器中自定义事件类型的数量、属性个数以及属性值的范围，流中的每个事件都是随机生成的。实验中该数据集包含了 100000 个原始事件。

第二个数据集包含了车辆交通数据，该数据由传感器收集，来自于丹麦奥胡斯市[23]。该数据集是通过传感器在 449 个观测点位收集了 4 个月的数据而得到的，总共包含 13577132 个原始事件。每个事件代表一个观察点的交通情况，一个事件的属性包括 ID、该点平均车速以及过去 5 分钟内观察到的车辆总数。

本文优化方法将与目前比较流行的基于有限状态自动机的 SASE 方法和 Siddhi 方法进行对比实验。其中本文提出的

基于临时匹配链式分区存储结构的匹配优化方法记为 LinkedCEP，实验在 Intel Core 2.60 GHz CPU i7 和 16G 内存的 Linux 系统上进行。

4.2 实验分析

首先实验比较的是本文的 LinkedCEP 与 SASE、Siddhi 在不同数据集上的匹配性能和临时匹配产生的数量。由于本文提出的 CoverMatch 算法只会匹配事件流中的复杂事件实例覆盖，并不会产生所有的匹配结果，为了公平地进行匹配性能的比较，在 LinkedCEP 中将包含匹配结果联结算法 CombineMatch 以支持产生所有的匹配结果。

由于两个数据集的原始事件、事件属性存在差异，为了更好地在两个数据集上执行匹配，针对两个数据集分别合成了不同的查询，并在两个数据集上分别使用 LinkedCEP、SASE、Siddhi 来执行对应的查询。以下将设计两组实验进行对比分析。

实验一：对两个数据集各合成了 5 组查询，每组查询包含 10 个基本复杂事件查询，这些复杂事件查询的序列长度为 3，由于时间窗口对匹配时间的影响很大，所以时间窗口统一为 50s，对应的值则是该组查询的平均匹配时间。

图 6 与 7 所展示的是分别在 ABC 事件流和交通事件流上匹配性能的对比结果。在查询序列长度和事件窗口一致的情况下，无论在任一组查询上，LinkedCEP 的匹配效率都要优于其余两种方法。例如，对于图 6 中 Q_3 查询组的匹配结果来说，LinkedCEP 花费的时间为 220ms，而 SASE、Siddhi 所花费的时间分别为 385ms 和 290ms。同样在图 7 中，在处理交通事件流的查询匹配上，LinkedCEP 的匹配效率同样优于 SASE 和 Siddhi，例如对于 Q_2 查询组的匹配结果来说，LinkedCEP 花费的时间为 130ms，而 SASE、Siddhi 花费的时间分别为 240ms 和 160ms。

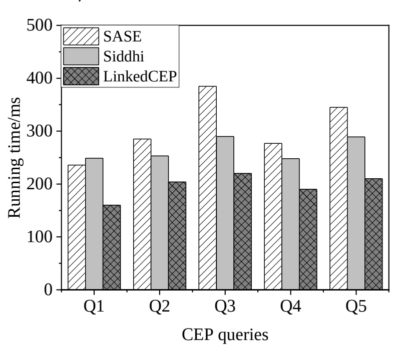


图 6 在 ABC 数据集上处理 5 组查询的性能对比

Fig. 6 Performance comparison of processing 5 groups of queries on the ABC data set

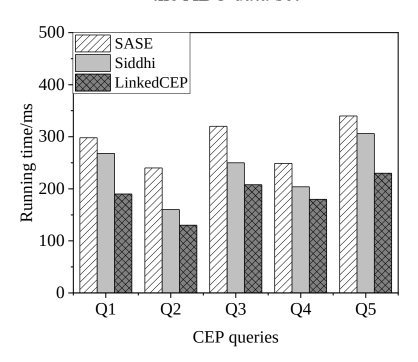


图 7 在交通数据集上处理 5 组查询的性能对比

Fig. 7 Performance comparison of processing 5 groups of queries on the traffic data set

实验二：测试在不同时间窗口下的匹配性能。使用查询一中的查询组，更改查询的时间窗口为 50s、100s、150s、200s、300s，对应的值为该组查询的平均匹配时间。

如图 8 和 9 所示，在不同的时间窗口下，LinkedCEP 的处理性能优于其余两个方法。例如图 8 的实验对比结果中，当时间窗口为 200s 时，LinkedCEP 的处理性能比 SASE 提升了 34%，比 Siddhi 提升了 19%。并且通过图 8 和 9 可以发现，随着时间窗口的增大，LinkedCEP 对比另外两个方法的性能提升更大，这是因为时间窗口的大小会影响临时匹配的数量，当时间窗口很大，堆积的临时匹配就多，对性能的影响也越大。

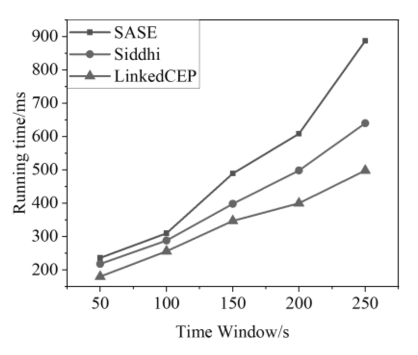


图 8 在 ABC 数据集上时间窗口约束的大小对性能影响的实验对比

Fig. 8 Experimental comparison of the effect of the size of the time window constraint on the performance on the ABC data set

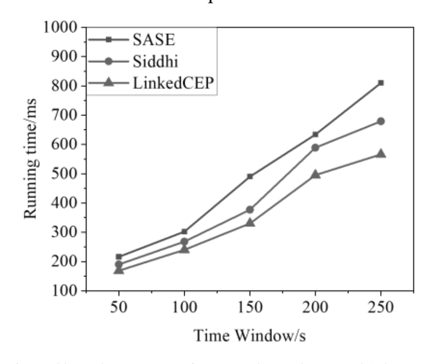


图 9 在交通数据集上时间窗口约束的大小对性能影响的实验对比

Fig. 9 Experimental comparison of the effect of the size of the time window constraint on the performance on the traffic data set

实验三：测试在处理不同的查询规模时的事件吞吐量。事件吞吐量指的是方法每秒处理的事件个数，吞吐量越大，说明方法的计算效率越高。实验中为 ABC 数据集和交通数据集分别生成了五组查询，其中每组查询中的查询数量依次为 50、100、150、200、250，每个查询的时间窗口都固定为 50s。实验对 SASE、Siddhi、CoverMatch 以及 LinkedCEP 分别展开了测试。

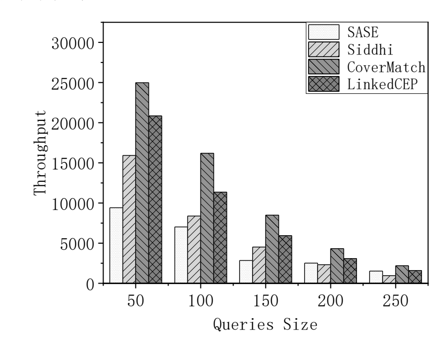


图 10 在 ABC 数据集上查询规模对吞吐量影响的实验对比

Fig. 10 Experimental comparison of the impact of query size on throughput on the ABC dataset

如图 10 与 11 所示，在两个数据集上，LinkedCEP 的吞吐量比 SASE 和 Siddhi 更大，而 CoverMatch 由于只匹配了复杂事件实例覆盖，LinkedCEP 是在 CoverMatch 的基础上进行匹配结果联结得到全部的匹配结果，所以 CoverMatch 的

吞吐量要比 LinkedCEP 更大。通过实验结果可以观察到，在交通数据集上进行复杂事件处理的吞吐量比在 ABC 数据集上进行相同工作时的吞吐量要小，这是由于 ABC 数据集在通过事件流生成器生成时采用了相同事件类型在一定概率上相邻的策略，以此来模拟出现复杂事件实例覆盖的场景，将会存在更多的匹配结果，而交通数据集中匹配结果较少，致使临时匹配由于无法产生最终匹配而在系统中一直停留，直到起始时间戳超出当前时间窗口范围才被淘汰，由此导致吞吐量降低。

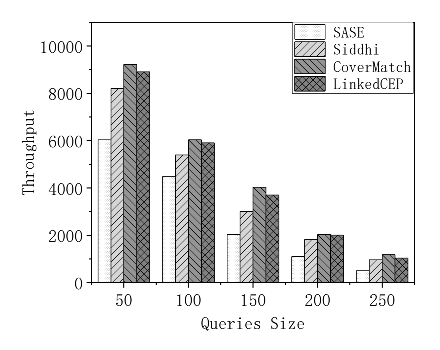


图 11　在交通数据集上查询规模对吞吐量影响的实验对比

Fig. 11　Experimental comparison of the impact of query size on throughput on the traffic dataset

在图 11 中，由于交通数据集中复杂事件实例匹配数量的减少，CoverMatch 方法和 LinkedCEP 方法的吞吐量大小接近。在 ABC 数据集中，CoverMatch 和 LinkedCEP 方法的吞吐量在五种查询规模的情况下都在 SASE 和 Siddhi 之上，并且比在交通数据集上提升更显著，这说明了当数据流中出现类型相同的事件相邻的情况时，本文提出的方法性能提升更加显著。

上述实验表明，本文提出的方法能够有效地提升基于有限状态自动机的复杂事件处理技术的匹配效率，无论是在查询序列长度较多还是查询时间窗口较大的情况下都能够通过减少冗余计算来获得效能的提升。

5　结束语

本文研究了基于有限状态自动机的复杂事件匹配优化问题。为了解决匹配过程中出现大量临时匹配所造成的低效匹配的问题，提出了复杂事件实例覆盖的概念。同时设计了一种基于临时匹配的链式分区存储结构，以及在该结构上进行高效匹配的方法，来利用复杂事件实例覆盖减少匹配过程中的冗余计算。实验结果表明了利用复杂事件实例覆盖的匹配技术所进行的匹配能够有效提升复杂事件匹配性能。

复杂事件处理技术具有广阔的应用空间，未来的研究工作将主要围绕多个复杂事件之间进行查询优化来展开。首先在多个查询上，设计一种算法来利用本文提出的临时匹配链式分区存储结构来共享复杂事件实例覆盖链，并探索并设计出一种适用于多查询进行结果共享的监测模型。最后进行模型的共享能力分析并通过实验来分析其处理性能。

参考文献：

[1] Demers A, Gehrke J, Hong M, *et al.* Towards expressive publish/subscribe systems [C]// International Conference on Extending Database Technology. Springer, Berlin, Heidelberg, 2006: 627-644.

[2] 产院东，郭乔进，梁中岩，等. 规则引擎发展综述 [J]. 信息化研究, 2021, 47 (2): 6. (Chan Yuandong, Gou Qiaojin, Liang Zhongyan, *et al.* A Survey of Rule Engine [J]. Informatization Research, 2021, 47 (2): 6.)

[3] Hill M, Campbell M, Chang Y C, *et al.* Event detection in sensor networks for modern oil fields [C]// Proceedings of the second international conference on Distributed event-based systems. 2008: 95-102.

[4] Zhou Q, Simmhan Y, Prasanna V. Incorporating semantic knowledge into dynamic data processing for smart power grids [C]// International Semantic Web Conference. Springer, Berlin, Heidelberg, 2012: 257-273.

[5] 赵会群，李会峰，刘金銮. RFID 物联网复杂事件模式聚类算法研究 [J]. 计算机应用研究, 2018, 35 (2): 3. (Zhao Huiqun, Li Huifeng, Liu Jinluan. Study on RFID complex event pattern clustering algorithm of Internet of things [J]. Application Research of Computers, 2018, 35 (2): 3.)

[6] Rahmani A M, Babaei Z, Souri A. Event-driven IoT architecture for data analysis of reliable healthcare application using complex event processing [J]. Cluster Computing, 2021, 24 (2): 1347-1360.

[7] 乔雅正，程良伦，王涛，等. 地铁列车环境中多依赖复杂事件处理研究 [J]. 计算机应用研究, 2019, 036 (008): 2355-2358, 2367. (Qiao Yazheng, Cheng Lianglun, Wang Tao, *et al.* Study on multi-dependency complex event processing in subway train environment [J]. Application Research of Computers, 2019, 036 (008): 2355-2358, 2367.)

[8] Agrawal J, Diao Y, Gyllstrom D, *et al.* Efficient pattern matching over event streams [C]// Proceedings of the 2008 ACM SIGMOD international conference on Management of data. 2008: 147-160.

[9] Zhang H, Diao Y, Immerman N. On complexity and optimization of expensive queries in complex event processing [C]// Proceedings of the 2014 ACM SIGMOD international conference on Management of data. 2014: 217-228.

[10] Demers A J, Gehrke J, Panda B, *et al.* Cayuga: A General Purpose Event Monitoring System [C]// Cidr. 2007, 7: 412-422.

[11] Brenna L, Demers A, Gehrke J, *et al.* Cayuga: a high-performance event processing engine [C]// Proceedings of the 2007 ACM SIGMOD international conference on Management of data. 2007: 1100-1102.

[12] Suhothayan S, Gajasinghe K, Loku Narangoda I, *et al.* Siddhi: A second look at complex event processing architectures [C]// Proceedings of the 2011 ACM workshop on Gateway computing environments. 2011: 43-50.

[13] Noh W F. CEL: A time-dependent, two-space-dimensional, coupled Eulerian-Lagrange code [R]. Lawrence Radiation Lab. , Univ. of California, Livermore, 1963.

[14] 王亦雄，廖湖声，孔祥翾，等. CEStream: 一种复杂事件流处理语言 [J]. 计算机科学, 2017, 44 (4): 5. (Wang Yixiong, Liao Husheng, Kong Xiangxuan, *et al.* CEStream: A Complex Event Stream Processing Language [J]. Computer Science, 2017, 44 (4): 5.)

[15] Diao Y, Immerman N, Gyllstrom D. Sase+: An agile language for kleene closure over event streams [J]. UMass Technical Report, 2007.

[16] Online Apache flinkcep [EB/OL]. [2020-12-04]. https://ci. apache. org/projects/flink/flink-docs-stable/dev/libs/cep. html.

[17] Mei Y, Madden S. Zstream: a cost-based query processor for adaptively detecting composite events [C]// Proceedings of the 2009 ACM SIGMOD International Conference on Management of data. 2009: 193-206.

[18] Fairoz Q. Kareem, Adnan Mohsin Abdulazeez, Dathar A. Hasan. Predicting Weather Forecasting State Based on Data Mining Classification Algorithms [J]. Asian Journal of Research in Computer Science, 2021.

[19] Patroumpas K, Alevizos E, Artikis A, *et al.* Online event recognition from moving vessel trajectories [J]. GeoInformatica, 2017, 21 (2): 389-427.

[20] Huang Qiuyang, Yang Yongjian, Xu Yuanbo, *et al.* Citywide road-network traffic monitoring using large-scale mobile signaling data [J]. Neurocomputing, 2021, 444.

[21] Zhang J. Multi-source remote sensing data fusion: status and trends [J]. International Journal of Image and Data Fusion, 2010, 1 (1): 5-24.

[22] 卢莉萍，张晓倩. 复杂环境下多传感器目标识别的数据融合方法 [J]. 西安电子科技大学学报, 2020, 47 (4): 8. (Lu Liping, Zhang Xiaoqian. Datafusion method of multi-sensor target recognition in complex enviroment [J]. Journal of Xidian University, 2020, 47 (4): 8.)

[23] Ali M I, Gao F, Mileo A. Citybench: A configurable benchmark to evaluate rsp engines using smart city datasets [C]// International Semantic Web Conference. Springer, Cham, 2015: 374-389.